**CheapSpeak**

1. **Introduction**

CheapSpeak est un logiciel d’audioconférence par voix sur IP (VoIP). Il permet de discuter à plusieurs dans des salons de discussions, avec d'autres utilisateurs, par la voix ou par messages textes.

L’application permet à l’utilisateur de choisir un pseudo avant de se connecter à un serveur. Il peut rejoindre un salon de discussion, parler, couper son micro, gérer le volume sonore et envoyer des messages textes aux personnes présentes dans le même salon.

1. **Architecture Client-Serveur**

L’architecture est orientée événements. Chaque action (comme une connexion/déconnexion) ou interaction (comme un envoie de message, ou un changement de salon) est un événement représenté en Java par la classe abstraite Event.

La communication client-serveur est ainsi réalisée par l’échange d’objet de la classe Event en utilisant le protocole TCP/IP.

Le diagramme ci-dessous illustre l’architecture globale du serveur.

Celui-ci est compose de plusieurs threads qui communiquent entre eux à l’aide de buffers partagés:

• **VocalServer Listening Thread**: Écoute le port pour toute nouvelle connexion. Lorsqu’un nouveau client se connecte, une nouvelle socket est créée, ainsi qu’un objet ClientConnection

• **ClientConnection Thread**: Stub qui permet la communication directe avec le client. Lorsqu’il reçoit un nouvel Event du client, il traite l’événement puis l’envoie au Broadcaster. Un objet ClientConnection est associé à chaque connexion.

• **Broadcaster Thread**: Se charge de redistribuer les événements aux clients concernés. Par exemple, il n’envoie les messages ou les paquets audio qu’aux clients présents dans le même salon que le client qui a produit le message.

La création de threads étant une opération lourde, la gestion des threads est gérée par le serveur à l’aide de deux Threadpools différentes: une FixedThreadPool et une CachedThreadpool.

La FixedThreadPool crée un nombre de threads (statiquement calculé à la compilation par la constante THREADS\_NUMBER) dès le démarrage du serveur.

La CachedThreadPool intervient lorsque tous les threads de la FixedThreadpool sont occupés et crée de nouveaux threads dynamiquement.

Pour éviter de recréer des threads inutilement, l’utilisation des threads de la FixedThreadpool est prioritaire sur la création de nouveaux par la CachedThreadpool.

Une image contenant capture d’écran

Description générée automatiquement

1. **Interface graphique**

L’application s’ouvre d’abord sur un splash screen le temps du chargement de l’application et de l’objet WindowMain. Notons bien qu’aucun temps de chargement n’est réellement nécessaire et que cet écran de démarrage est uniquement esthétique.

Toute l’interface graphique est gérée à partir de cette classe centrale, WindowMain, qui représente la fenêtre de l’application. C’est dans cette classe que les panneaux de connexion (PanelConnect), de discussion (PanelChat) et d’arborescence salons-utilisateurs (TreeRoom) vont s’afficher. Toutes ces classes vont hériter des objets de la bibliothèque javax.swing. La classe WindowMain contient aussi la barre de menu (MenuBar), et toutes les classes gérant les événements (avec la bibliothèque java.awt) pour que l’utilisateur puisse interagir avec l’application, comme se connecter ou envoyer un message.

Il est aussi possible de changer le thème de l’interface graphique dans les menus (light ou dark). Cela est géré par les classes UIManager et UIResources.

Enfin, WindowMain implémente l’interface EventEngine permettant de gérer les événements (Event) que le serveur envoie au client. Ces événements sont gérés dans une boucle dans la méthode eventListener(). Afin de ne pas avoir à recréer un thread à chaque déconnexion pour écouter le serveur, cette boucle est mise en attente par une variable conditionnelle. Ce « monitor pattern » est aussi utilisé pour couper le microphone dans la gestion du son.

1. **Gestion du son**

Toute la gestion du son est effectuée par l’objet AudioProcessor (cet objet est instancié dans la classe WindowMain). Celui-ci permet d’écouter le micro de l’utilisateur avec la classe Microphone. Il est bien-sûr possible de désactiver/réactiver le micro dans les menus (mute/unmute).

De plus, chaque flux sonore des autres utilisateurs présents dans le même salon est géré dans un objet AudioChannel différent contenant chacun un objet Speaker pour lire les SoundPacket (objets portant le signal sonore) envoyés par le serveur. Chacun de ces AudioChannel est associé dans une Hashtable à un utilisateur. Notons que la classe AudioChannel hérite de la classe Thread, ce qui permet d’avoir plusieurs utilisateurs parlant en même temps.

Lorsqu’un SoundPacket contient un signal sonore trop faible (c’est-à-dire que l’utilisateur ne parle pas), celui-ci est quand même envoyé au serveur avec un bruit à la place pour éviter les sauts de volume lorsqu’un utilisateur se met à parler. C’est pourquoi on entend toujours un petit grésillement lorsque le micro d’un autre utilisateur est allumé.

Enfin, la classe AudioProcessor contient aussi un thread pour retenir quel utilisateur est en train de parler et ainsi pouvoir afficher un icône haut-parleur à côté de son pseudo dans l’arborescence lorsqu’il parle.