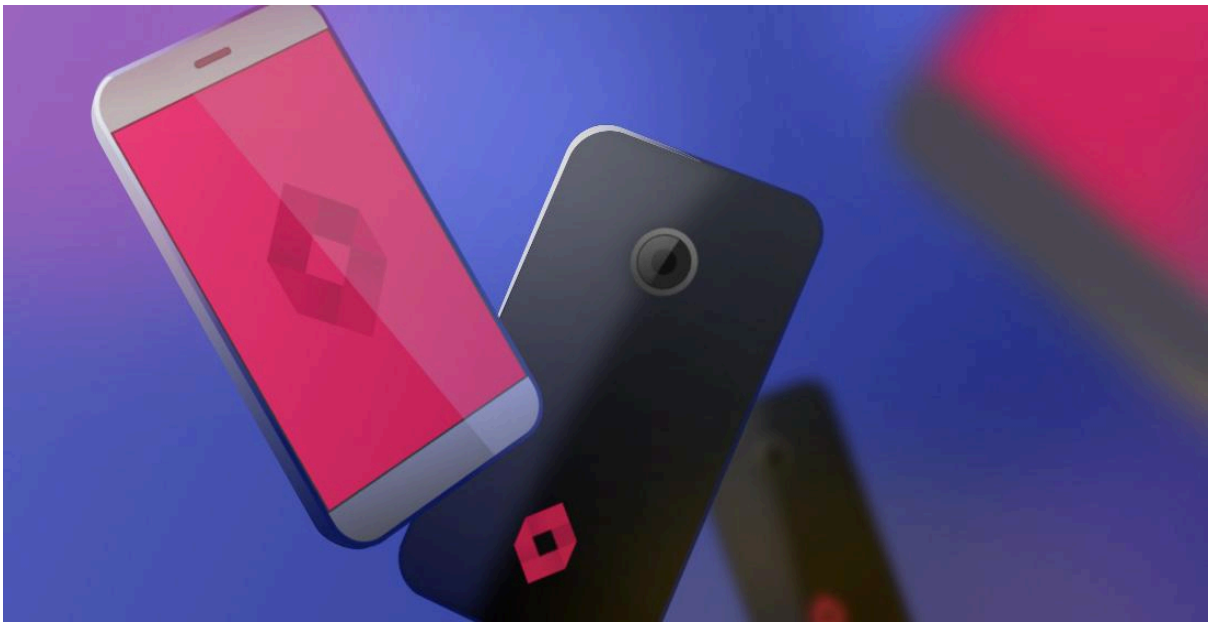
3il Ingénieur - UML

Document de conception

Aide à la communication pour personnes malentendantes

Jodie Monterde & Romain Courbaize & Maxime Froissant & Gabriel Monczewski

---



Le document de conception a pour but de présenter une version prototype de l'application. Pour ce faire, nous avons choisi un incrément défini, pour nous limiter à une partie de l'application. Puis, à partir de cet incrément, nous avons conceptualisé différents diagrammes UML pour nous aider dans le développement par la suite.

Architecture logique du logiciel

1. Description globale

Le système se compose des modules principaux suivants :

- **Module de transcription :** Responsable de la reconnaissance vocale et de la génération en temps réel de textes transcrits.
- **Module de détection sonore :** Capable de détecter des sons spécifiques (alarme, appel de nom) dans l'environnement.
- **Module de notifications :** Gère les alertes visuelles et les notifications pour l'utilisateur via l'interface utilisateur.
- **Interface utilisateur (UI) :** Présente les transcriptions et les alertes à l'utilisateur sous une forme claire et accessible.
- **Module central de coordination :** Sert de médiateur entre les modules et assure leur synchronisation.

2. Interaction entre les composants

1. **Le module de transcription** reçoit un flux audio en entrée, le traite à l'aide d'un service de reconnaissance vocale et transmet les résultats au **module central de coordination**.
2. **Le module de détection sonore** analyse le même flux audio pour identifier des sons spécifiques. Lorsqu'un son pertinent est détecté, il envoie une notification au **Module central de coordination**.
3. Le **Module central de coordination** transmet les données (texte transcrit et notifications de sons détectés) au **Module de notifications** et à l'**Interface utilisateur**.
4. L'**Interface utilisateur** affiche les transcriptions en temps réel et présente des alertes visuelles adaptées.

3. Technologies envisagées

- **Reconnaissance vocale :** API Google Speech-to-Text (Google Cloud) ou une solution open-source comme CMU Sphinx.

Description de l'incrément choisi

Afin de présenter une version prototype viable permettant au client de voir le fonctionnement de notre application, nous avons décidé, dans un premier incrément, de réaliser la fonctionnalité phare de l'application : la transcription en temps réel de la parole.

Pour rappeler le principe : La transcription en temps réel de la parole a pour objectif de permettre à l'utilisateur malentendant de suivre facilement une conversation en direct grâce à une interface visuelle. Cependant, cette transcription n'est pas toujours nécessaire et peut être activée ou désactivée à tout moment en fonction des besoins de l'utilisateur.

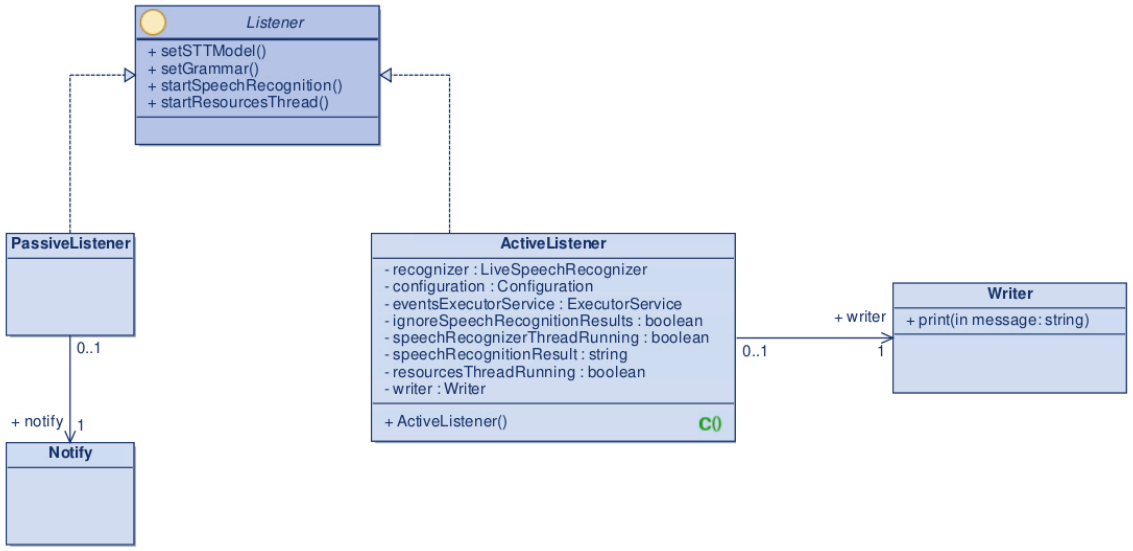
Lorsqu'elle est activée, le système capte les voix des participants via un microphone intégré ou externe. Il analyse ensuite le flux audio à l'aide d'un moteur de reconnaissance vocale. Ce moteur convertit les paroles en texte, qui sera ensuite affiché instantanément sur un écran. Depuis l'écran de l'appareil, l'utilisateur pourra lire et donc suivre tout ce qui est dit. Pour le moment, le système analysera uniquement la langue française, la compréhension multi-linguiste pouvant être un axe d'amélioration lors d'une prochaine mise à jour.

Cette approche évite une consommation inutile de ressource lorsque la transcription n'est pas requise.

Conception détaillée

Pour l'incrément choisi, les étudiants fourniront un diagramme de classes, des diagrammes de séquence, des diagrammes d'états-transitions, et éventuellement, un diagramme d'objets. Un soin particulier devra être apporté à la cohérence entre ces diagrammes.

Diagramme de classe

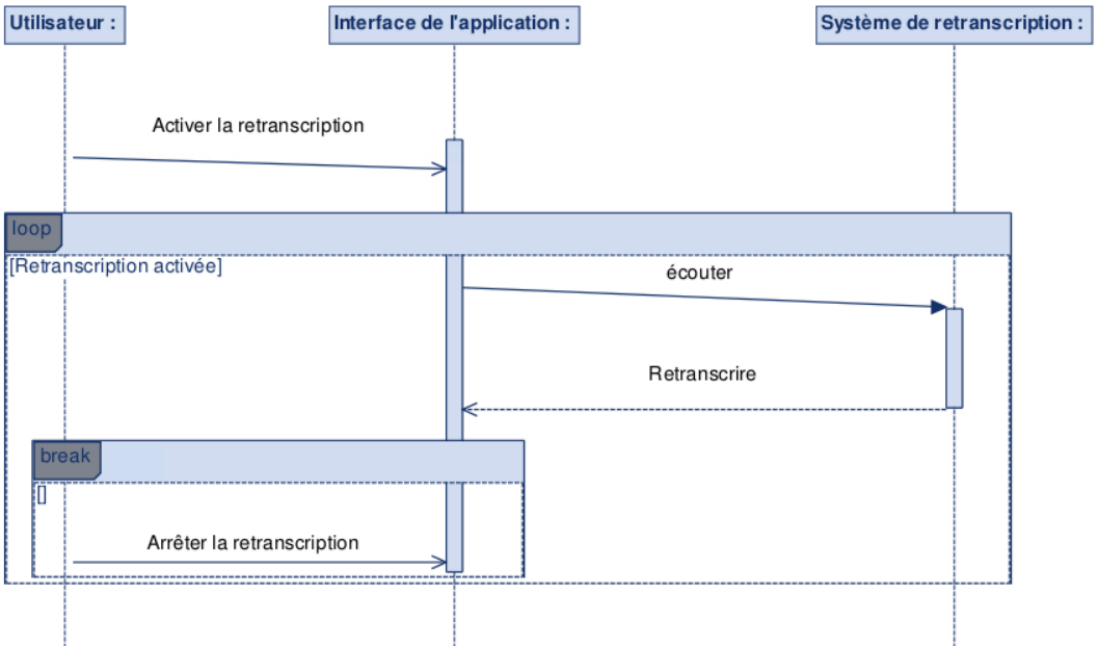


La classe **Notify** a pour responsabilité l'envoie de notification. La classe **PassiveListener** écoute les sons et lorsqu'elle détecte un son, elle transmet l'information à la classe **Notify**. La classe **ActiveListener** est utilisée pour écouter et comprendre le son ambiant. Elle transmet le texte à la classe **Writer** qui elle va afficher le résultat.
Les classes **PassiveListener** et **ActiveListener** implémentent l'interface **Listener** car elles ont toutes les deux la responsabilité d'écouter.

Les classes **LiveSpeechRecognizer**, **Configuration** et **ExecutorService** sont des classes utilisées par la librairie Sphinx4 afin d'implémenter la retranscription vocale.

Diagramme de séquence

Ce diagramme de séquence représente le **cas de la retranscription en temps réel** de la parole d'un interlocuteur.

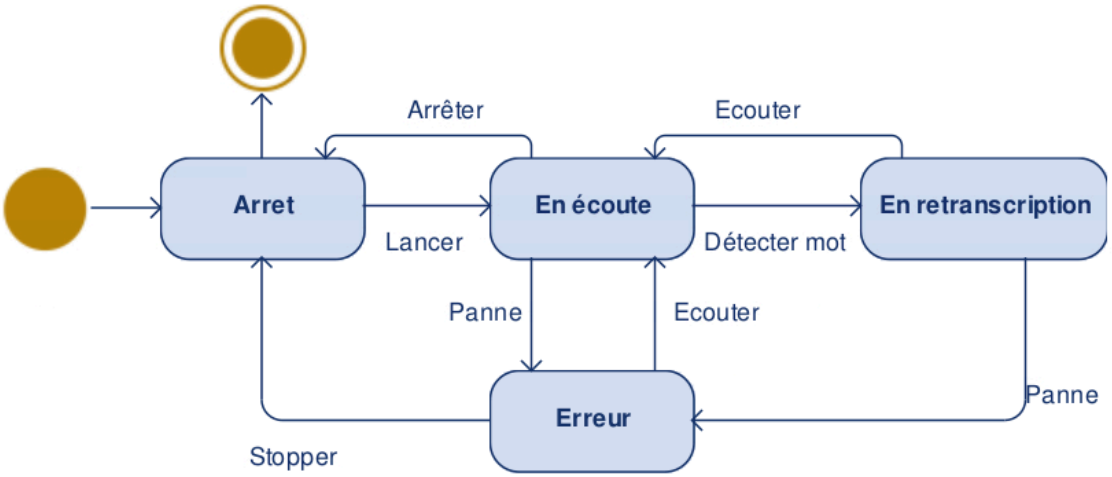


1. **Acteurs principaux :**
    - **Utilisateur** : Personne malentendante qui active ou désactive la retranscription via l'interface de l'application.
    - **Interface de l'application** : Le composant de l'application qui gère les interactions utilisateur et communique avec le système de retranscription.
    - **Système de retranscription** : Module chargé d'écouter les sons ou les paroles et de les transcrire en texte.
2. **Flux principal :**
    - **Activation de la retranscription** : L'utilisateur initie le processus en activant la retranscription à partir de l'interface de l'application.
    - Une boucle s'active tant que l'état "retranscription activée" est vrai.
3. **Traitement dans la boucle :**

- L'interface de l'application envoie une requête au système de retranscription pour "écouter".
- Le système de retranscription analyse les sons ou les paroles captées et renvoie le texte transcrit à l'interface.
- Cette boucle continue tant que l'utilisateur ne décide pas d'arrêter la retranscription.

4. **Interruption du processus :**
    - Lorsque l'utilisateur choisit d'arrêter la retranscription, il envoie une commande correspondante via l'interface de l'application.
    - Cela interrompt la boucle ("break") et le processus de retranscription s'arrête.

Diagramme d’états-transitions



Ce diagramme d’états-transitions représente le **système de retranscription** de notre application.

Arrêt

Cet état, qui est à la fois l’état **initial** et l’état **final**, fait référence à l’**arrêt total** de la retranscription. De cet état, on ne peut lancer le système de retranscription, et par conséquent devenir ‘en écoute’.

En écoute

Lorsque le système est ‘en écoute’, il **écoute activement** son environnement à la recherche de mots à retranscrire sous forme de texte. D’ici, le processus logique est donc de détecter un mot, ce qui nous fait passer à l’état ‘en transcription’. Cependant, rien ne nous empêche d’arrêter l’écoute. Le système peut également détecter une erreur lors de l’écoute.

En retranscription

A chaque fois qu’un mot est détecté lors de l’écoute, on passe à l’état ‘en retranscription’, ou celui-ci est traduit sous forme de texte. Puis le système repasse ‘en écoute’, détecte le mot suivant, le traduit, etc. On considère qu’on ne peut pas éteindre le système d’écoute

directement depuis l'état 'en retranscription'. C'est un état très court, qui oscille continuellement avec l'état 'en écoute'. On a donc considéré que l'arrêt ne se fait que depuis l'état 'en écoute'. Dans des conditions parfaites, on **reboucle** donc entre l'état 'en retranscription' et 'en écoute' jusqu'à tant que l'utilisateur souhaite arrêter l'écoute. Cependant, des erreurs peuvent survenir…

Erreur

Une erreur peut survenir soit durant la phase d'écoute, soit durant la phase de retranscription. Dans les deux cas, l'écoute ou la retranscription s'interrompent, un message d'erreur apparaît, et laisse la possibilité à l'utilisateur soit de reprendre l'écoute, soit d'arrêter le système.

Exemple : l'utilisateur lance le système de retranscription dans un environnement bruyant, ce qui empêche le système de fonctionner correctement. Un message d'erreur apparaît : 'Impossible de retranscrire dans ces conditions. Essayez de vous isoler dans un endroit plus calme'. Avec deux options : réessayer (ce qui relance la retranscription) ou arrêter (qui arrête celle-ci).

En retranscription, nous avons identifié moins d'erreurs, mais celles-ci peuvent toujours survenir. Cela fonctionne alors de la même manière.