Documentation interne du code

Synthèse vocale de numéros téléphoniques

Equipe:

CUMMINGS Thibaud
GOLETTO Michael
SOLEYMANKHANI Hossein
DIALLO Négué

Table de Révision

Révision	Date	Auteur	Modifications apportées
1	10 Avril 2010	N. DIALLO	Version initiale
2	26/27 Avril 2010	groupe	MAJ de 3. Documentation des modules
3	28 Avril 2010	T. CUMMINGS	Version finale

Table des matières

1 INTRODUCTION

Ce document apporte une explication globale de la librairie.

L'architecture de la librairie ainsi qu'une description succincte de chaque module figure dans ce document.

La documentation du code généré par *doxygen* complète ce document en apportant des détails techniques.

1.1 Objectifs et méthodes

Il est demandé de livrer une librairie codé en langage C permettant la synthèse vocale de numéro téléphoniques ainsi qu'une application qui teste cette librairie. La librairie qu'il est demandé de fournir doit comporter une fonction permettant la reconstruction vocale d'un numéro téléphonique. Afin de restituer vocalement des numéros de téléphones, il est nécessaire d'implémenter un algorithme de synthèse vocale. A partir d'une série de nombres en entrée, l'algorithme devra être capable de générer le fichier son correspondant à ces nombres par une voix reconstituée.

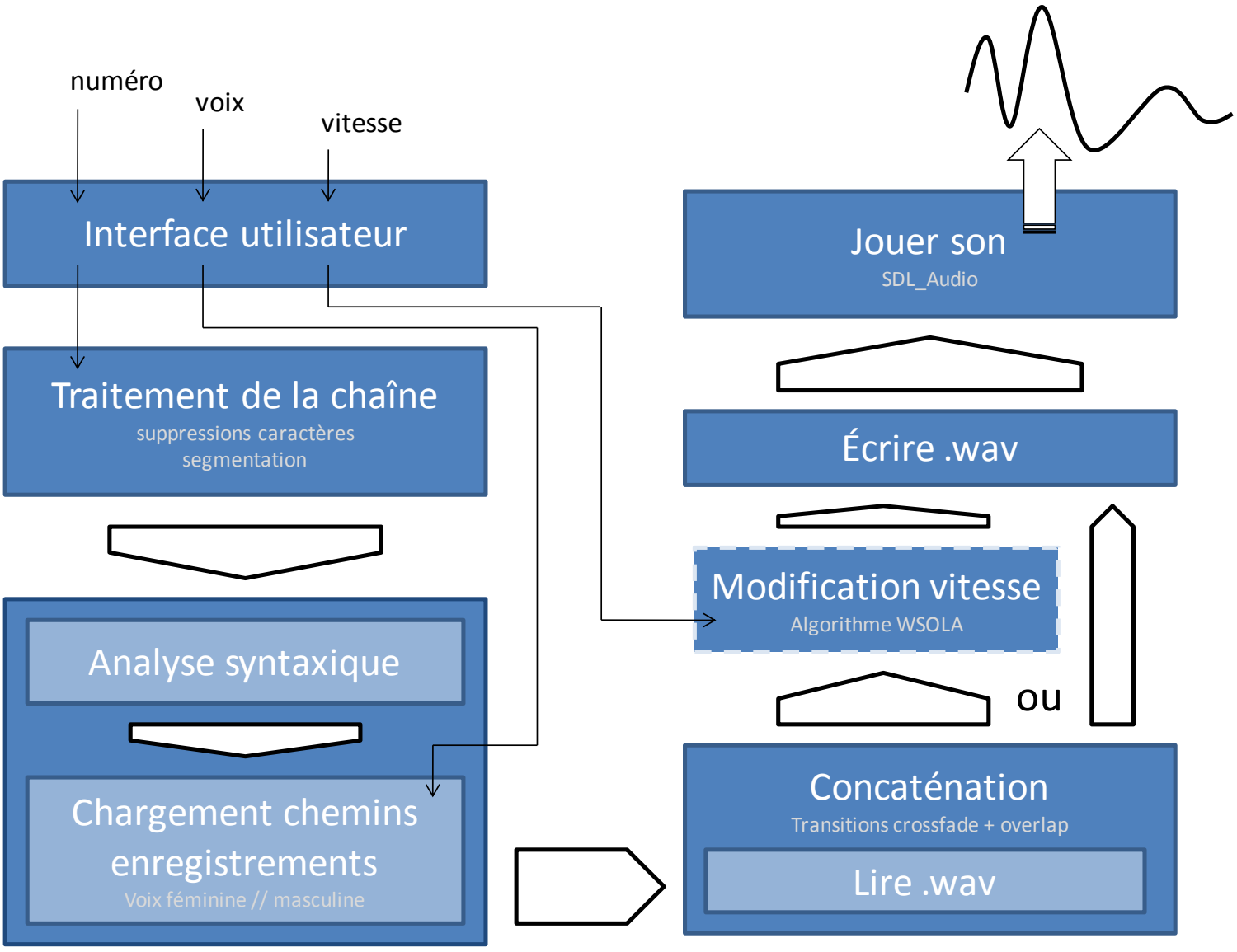
1.2 Documents de référence

Le deux documents de référence ici sont le cahier des charges (*S1*) et le cahier de recette (*S2*)

2 ARCHITECTURE

Voici un schéma illustrant l'architecture de la librairie.

Chaque rectangle bleu foncé correspond à un module, chaque rectangle bleu ciel correspond à un sous-module.



3 DOCUMENTATION DES MODULES

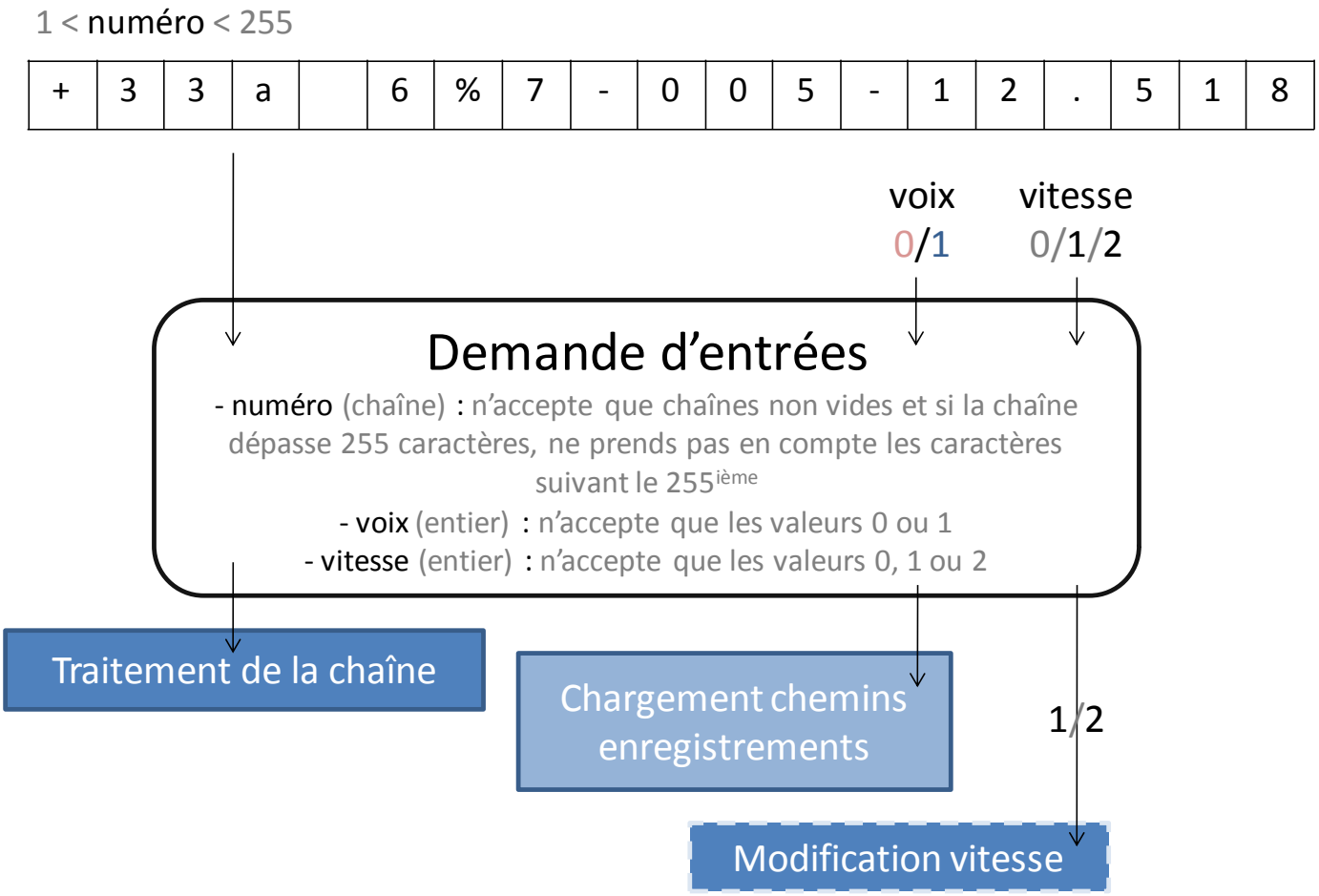
Chaque module est synthétiquement expliqué et illustré par des schémas.

3.1 Interface utilisateur

Le module interface utilisateur consiste à saisir les variables et à s'assurer que la saisie de celles-ci est effectuée correctement.
Celui ci demande d'entrer les trois variables : numéro, voix et vitesse.

- La chaîne de caractères correspondant au numéro ne peut pas être vide et si celle-ci dépasse 255 caractères, on ne doit pas tenir compte des caractères suivant le 255ième.
- L'entier correspondant à la voix désirée ne peut valoir autre que 0 ou 1, toute autre entrée ne doit pas être acceptée.
- L'entier correspondant à la vitesse ne peut valoir autre que 0, 1 ou 2, toute autre entrée ne doit pas être acceptée.

Voici un schéma illustrant le fonctionnement du module:

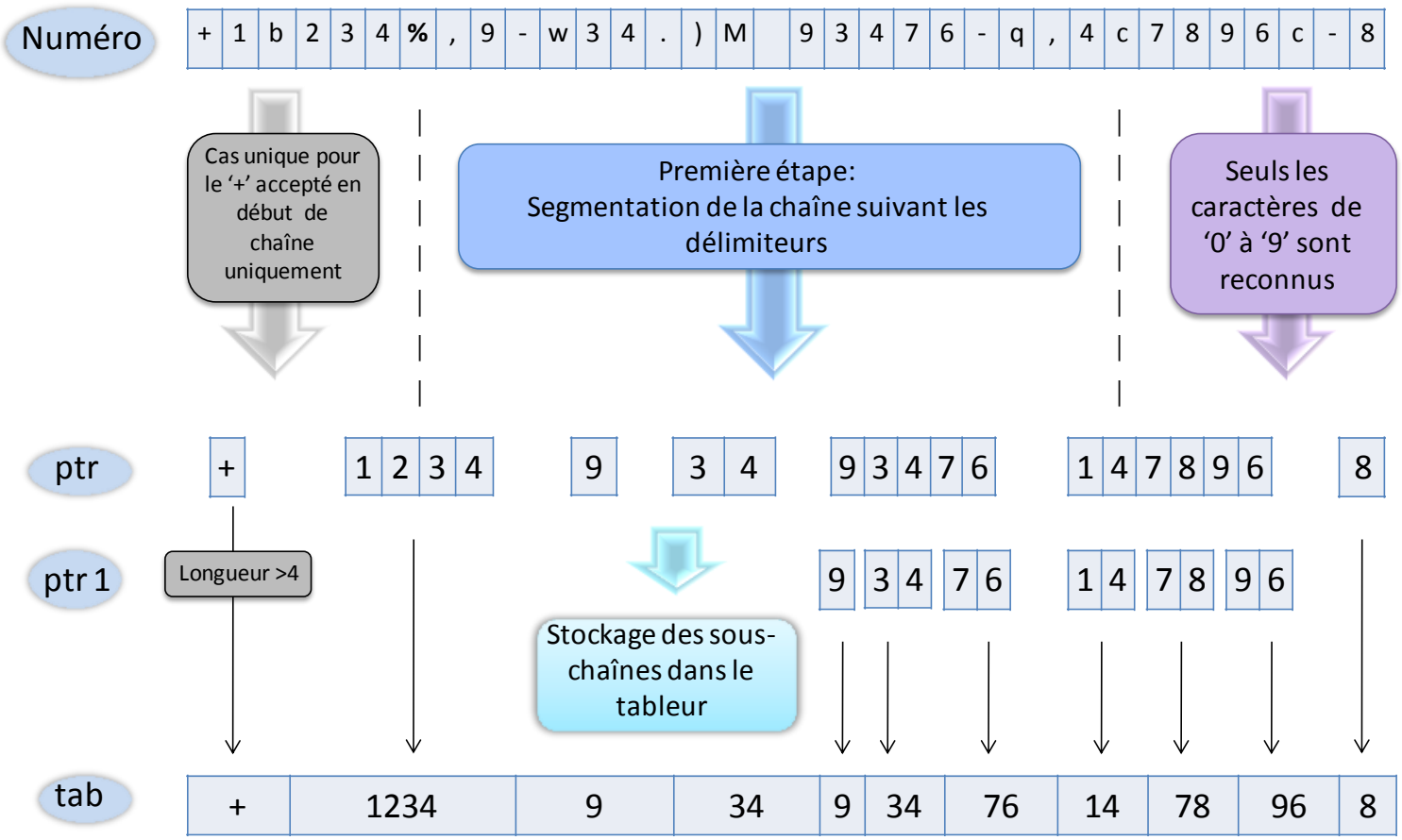


3.2 Traitement de la chaîne

Le module traitement de la chaîne consiste à filtrer et à segmenter la chaine correspondant au numéro de téléphone. La chaine est limitée à 255 caractères (ANSI) et n'accepte que les chiffres de 0 à 9 et cinq autres caractères : le point « . », la virgule « , », l'espace « », le tiret « - » et le signe mathématique plus « + ». Tout autre caractère sera ignoré.

- Le '+' est reconnu uniquement en début de chaine. Si il est retrouvé à tout autre position, il est supprimé.
- La chaîne est découpée en plusieurs segments (ne dépassant pas quatre caractères) délimités par les séparateurs : le point, la virgule, l'espace et le tiret.
- S'il n'y a pas de séparateurs, et que la chaîne est composé de plus de quatre caractères, la chaîne est segmentée par paire. Ces segments sont ensuite stockés dans le tableau de chaines.

Ci-dessous, un schéma qui illustre le fonctionnement du module:



3.3 Analyse syntaxique

3.3.1 Analyse syntaxique

Le module d'analyse syntaxique consiste à décomposer le numéro entré en chiffres ou numéros qui le constituent. exemple: 5243 => "cinq-mille-deux-cent-quarante-trois".

Pour un algorithme optimisé, il a été recherché un nombre minimum d'enregistrements possibles pour reconstruire tous les numéros de 0 à 9999. De cette manière, les enregistrements prendrait moins de place sur le disque dûr.

Dans la langue française les numéros entrés ne peuvent pas toujours se décomposer exclusivement en chiffres ou numéros strictement énumérés.
Par exemple: 8697 se prononce "hui-mille-si-cent-quatreuh-vingt-di-sept". En effet on s'aperçoit avec cet exemple que ce numéro ne se prononce pas "huit-mille-six-cent-quatre-vingt-dix-sept".
Il nous a donc fallu enregistrer d'autres "sons" tel que "hui" ou "di".

Le module utilise alors uniquement 33 enregistrements (33 pour chaque voix, soit 66 en tout) :

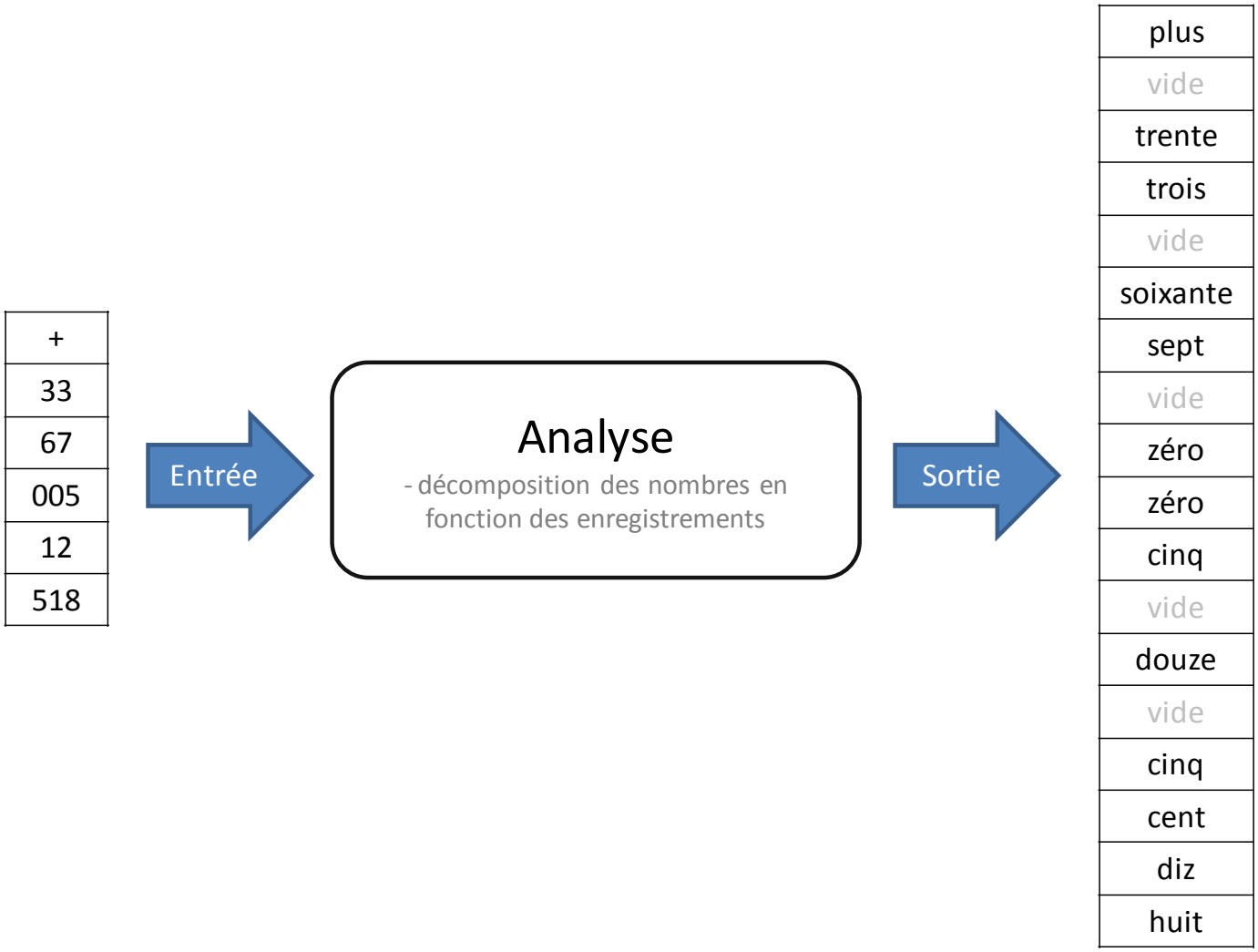
- 24 typiques:
  - "zéro" à "seize"
  - "vingt", "trente", "quarante", "cinquante", "soixante", "cent", "mille"

- 9 spécifiques:
  - "di" (pour 17, "di-sept"), "diz" (pour 18 et 19, "diz-huit")
  - "et" (pour 31, 41, 51; "trente-et-un"...), "vingtt" (pour 21 à 29; "vingt-t-cinq")
  - "quatree" (pour 80 à 99; "quatreuh-vingt")
  - "si" et "hui" (pour 600, 6000, 800, 8000; "si-cent", "hui-mille")
  - "vide" (pour marquer la séparation entre chaque numéros)
  - "plus" (pour +)

Le module prend donc en entrée un tableau de chaine de caractères correspondant aux numéros à synthétiser/jouer. Chaque case du tableau est analysée et un nouveau tableau de chaîne est construit.

Ce nouveau tableau (de sortie) construit est composé des noms des sons/enregistrements que composent les numéros de chaque case du tableau d'entrée.

Le schéma suivant illustre le module:

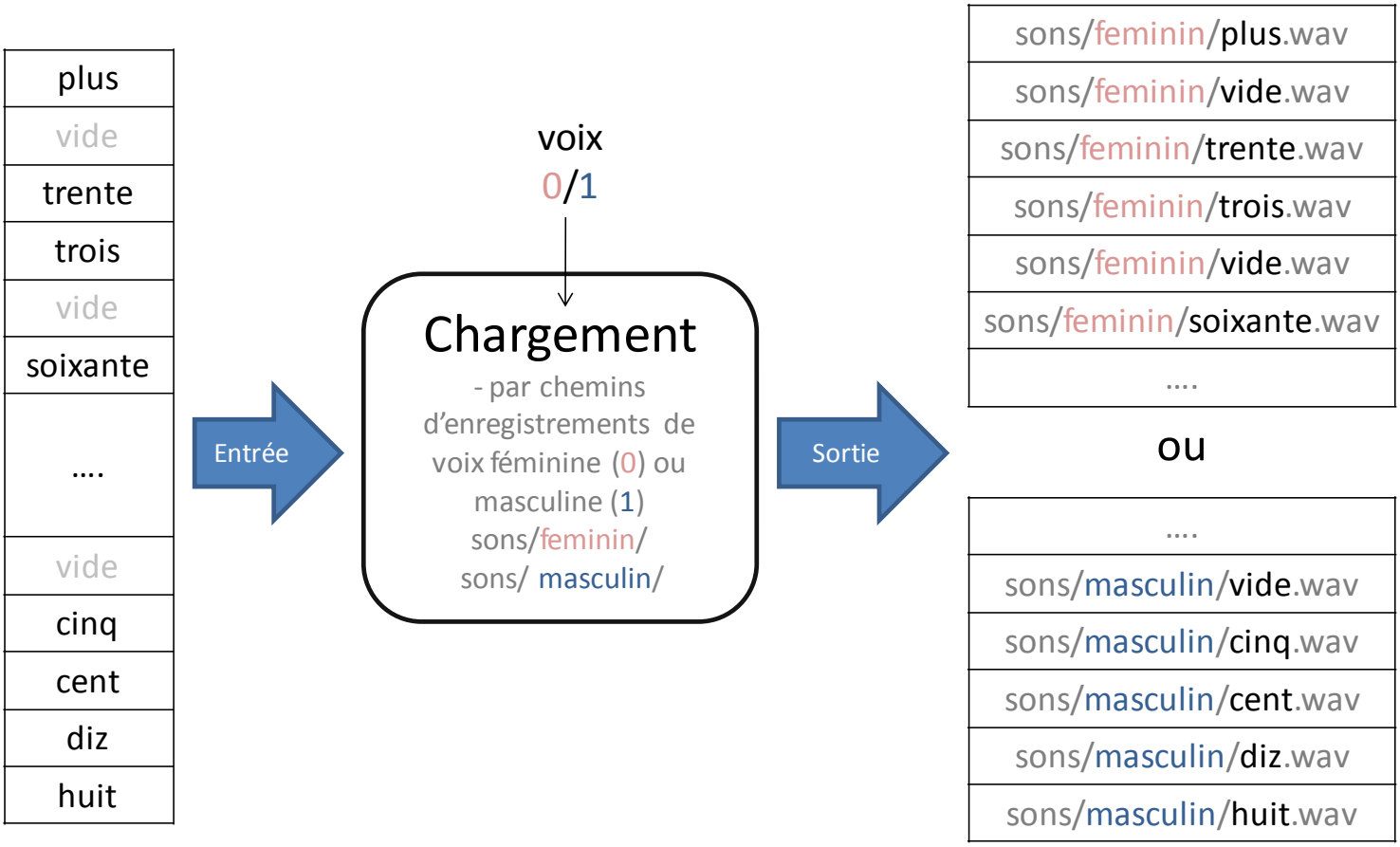


3.3.2 Chargements des chemins des enregistrements

Le sous-module de chargement des chemins d'enregistrements prend en entré le tableau de chaîne rempli par les numéros/sons à jouer et le modifie de sorte à avoir les chemins vers les enregistrements de ces numéros/sons (fichiers sons *.wav*)

En fonction de la variable voix, les chemins chargés correspondront à ceux des enregistrements masculins ou féminins.
Si voix vaut 0, les chemins chargés seront ceux des enregistrements de voix féminine; si voix vaut 1 les chemins chargés seront ceux des enregistrements de voix masculine.

Ci-dessous un schéma qui illustre le sous-module:



3.4 Lire/écrire fichier wave

Une structure WAVE a été créée à partir de la documentation officielle (Canonical WAVE file format). Elle permet de stocker toutes les données constituant un fichier son *.wav*.

Le module de lecture de fichier wave lit un fichier son *.wav* et stocke toutes ses données dans une structure WAVE.

Le module d'écriture d'un fichier *.wav* lit une structure WAVE et reconstitue à partir de ses données un fichier son *.wav*.

3.5 Concaténation

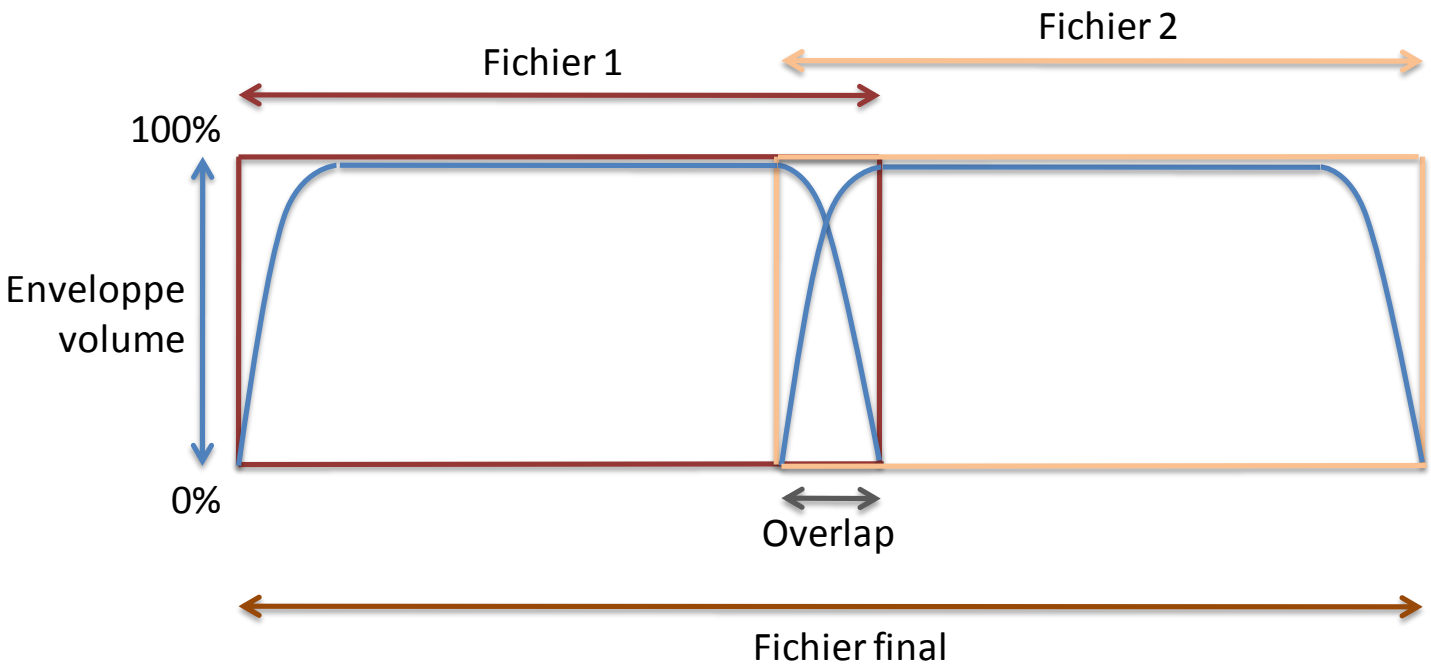
Ce module enregistre dans des structures adéquates les données des différents fichiers *WAVE*. Il effectue ensuite un amortissement sinusoïdale et un fondu avant de concaténer l'ensemble des données de ces structures dans une seule et unique structure.

Le chargement des structures est effectué par la fonction store, qui fait appel au module de lecture d'un fichier *WAVE*.

Pour chaque fichier, un amortissement sinusoïdal est effectué sur les échantillons des 10 premières ou dernières millisecondes.

Lors de la concaténation, on effectue sur ce même intervalle un chevauchement afin de rendre plus fluide l'écoute finale.

Le schéma suivant illustre le module:

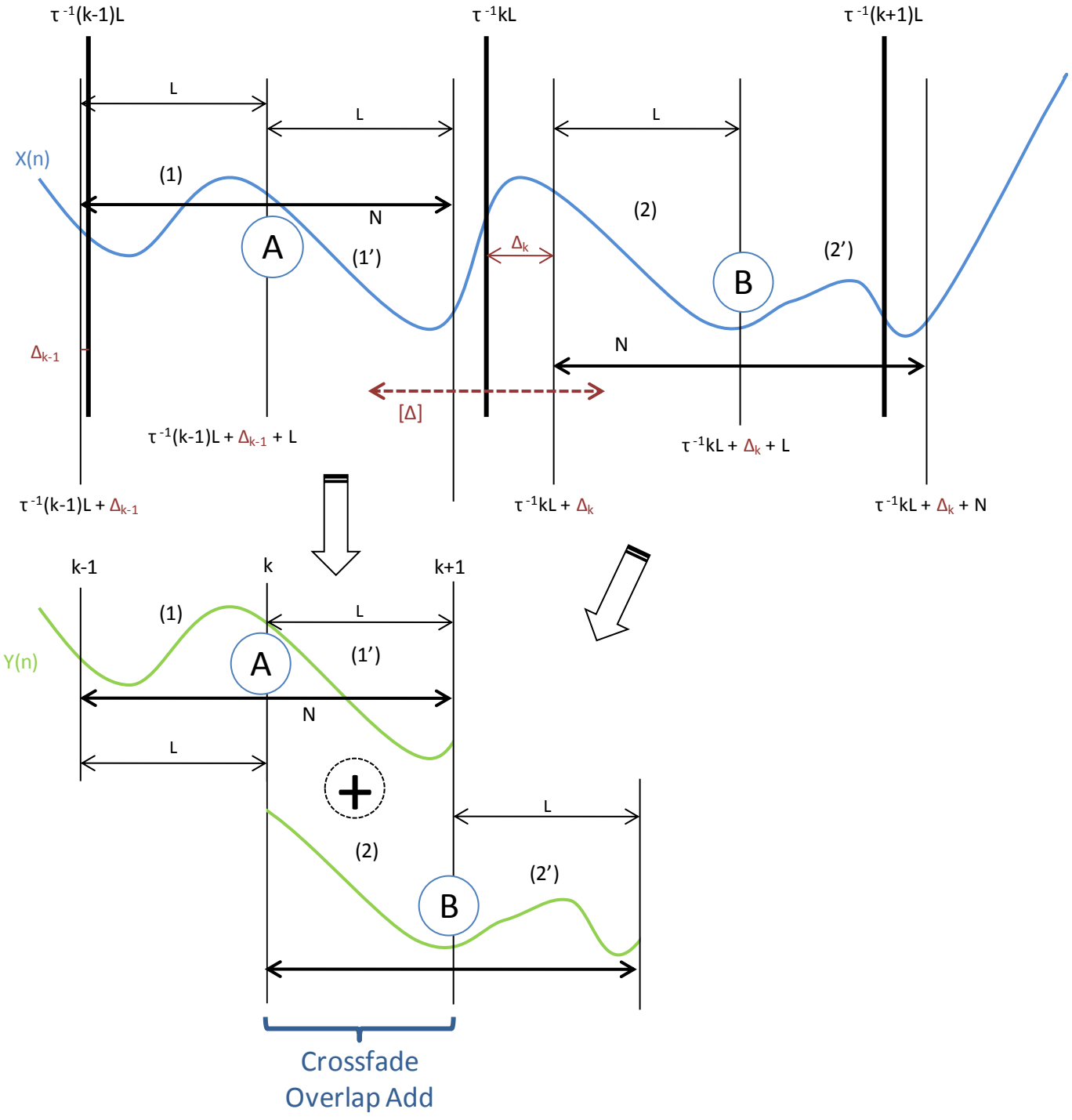


3.6 Modification vitesse (WSOLA)

Le module de modification de vitesse modifie une structure d'un fichier son *.wav* de telle sorte que la structure corresponde à un fichier son soit plus long, soit plus court.

L'algorithme utilisé est appelé WSOLA (*Wave-form Similarity based Over-Lap Add*). Cet algorithme créé une version modifiée (soit étendue soit rétrécie) de l'onde originale d'un son en concaténant en permanence des segments de temps court, ici d'une durée de 20 ms (soit N échantillons), extraits de l'onde originale.

Le schéma ci-dessous illustre le principe de l'algorithme à moment k:



τ est le facteur temps.

τ < 1 correspond à une augmentation de vitesse. En effet, si τ est <1 on a τ^{-1} >1 donc les seront plus espacés les uns des autres, soit les fenêtres seront plus grandes.

Il y aura donc des "suppressions" de segments. Le fichier généré en sortie sera alors rétrécit et donc la lecture de ce fichier se complètera plus rapidement; simulant une augmentation de vitesse de lecture.

Inversement τ > 1 correspond à une diminution de vitesse, si τ est >1 on a τ^{-1} <1 donc les seront moins espacés les uns des autre, soit les fenêtres seront plus petites.

Il y aura alors des "ajouts" de segments. Le fichier généré en sortie sera alors allongé et donc la lecture se complètera plus lentement; simulant donc une diminution de vitesse de lecture.

N correspond à un nombre d'échantillons fixe contenu dans 20ms; et N = 2L.

Le segment A [$\tau^{-1}(k-1)L + \Delta_{k-1}$ $\tau^{-1}(k-1)L + \Delta_{k-1} + N$] de taille N correspond au segment (1) [$\tau^{-1}(k-1)L + \Delta_{k-1}$ $\tau^{-1}(k-1)L + \Delta_{k-1} + L$] auquel on ajoute le segment qui suit, c'est-à-dire le segment (1') [$\tau^{-1}(k-1)L + \Delta_{k-1} + L$ $\tau^{-1}(k-1)L + \Delta_{k-1} + N$]. On a A = (1) + (1').

De même le segment B [$\tau^{-1}kL + \Delta_k$ $\tau^{-1}kL + \Delta_k + N$] de taille N correspond au segment (2) [$\tau^{-1}kL + \Delta_k$ $\tau^{-1}kL + \Delta_k + L$] auquel on ajoute le segment qui suit, c'est-à-dire le segment (2') [$\tau^{-1}kL + \Delta_k + L$ $\tau^{-1}kL + \Delta_k + N$]. B = (2) + (2').

Le schéma décrit une situation avec τ < 1 soit un rétrécissement de fichier, soit une augmentation de vitesse.

Se référant à la figure ci-dessus, lors de la *k-1*^{ième} itération de l'algorithme, le segment A obtenu à partir de X(n), est annexé à Y(n) à partir de la position k-1.

Puis, lors de la *k*^{ième} itération de l'algorithme; un segment de taille L est recherché dans la fenêtre [$\tau^{-1}kL$ $\tau^{-1}(k+1)L$] de X(n), de sorte que le segment recherché soit au maximum équivalent au segment qui suit le segment (1), c'est-à-dire au segment (1').

Le Δ correspond à un intervalle autour de chaque positions $\tau^{-1}kL$. Pour chaque Δ de l'intervalle, les L échantillons du segment (1') sont comparés aux L échantillons de chaque segment (2) de l'intervalle de Δ. Le calcul de similitude est effectué en faisant la somme des valeurs absolues de la différence entre les L échantillons des segments (1') et (2).

Une fois le segment maximalement similaire trouvé, c'est-à-dire le segment (2), le segment B est ajouté à Y(n) à partir de la position k.

Seulement à la position k il y a déjà le segment (1'). Il nous faut donc "fusioner" le segment (1) et (2) et pour cela la technique de "crossfade" est utilisé. Dans la fenêtre [k k+1], le volume (soit l'amplitude des échantillons) du segment (1') est diminué graduellement jusqu'à 0 et le volume du segment (2) est graduellement augmenté de 0 à sa valeur normale. Ce "crossfade" s'assure que cette "fusion" soit "lisse".

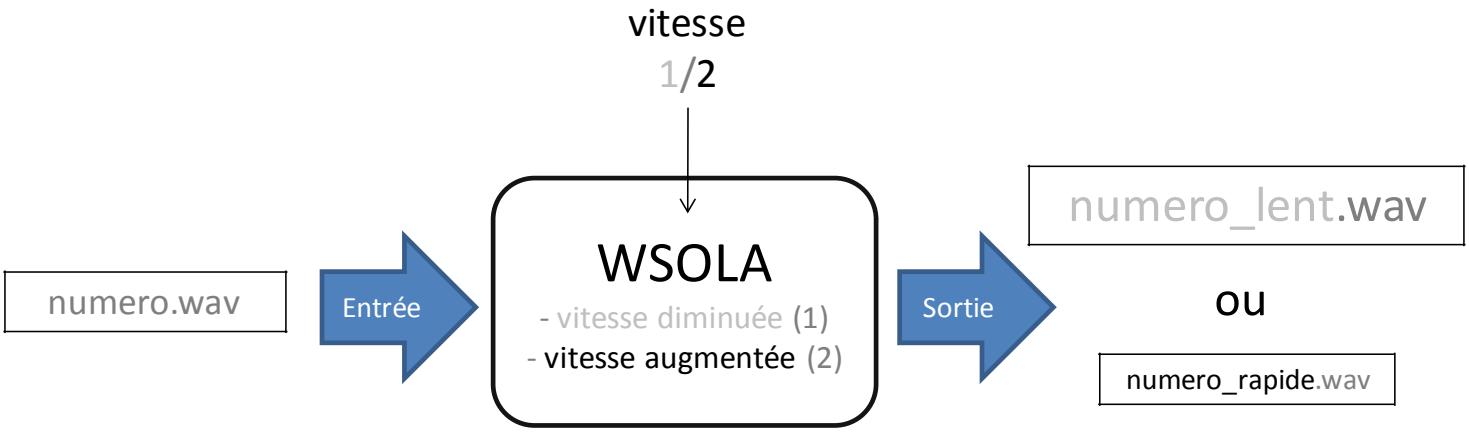
On peut donc observer la suppression de segment; en effet le segment [$\tau^{-1}(k-1)L + \Delta_{k-1} + N$ $\tau^{-1}(k-1)L + \Delta_{k-1} + L$] est supprimé; alors que les segments (1') et (2) ne font plus qu'un.

L'algorithme se poursuit ensuite par la recherche d'un segment qui est semblable au maximum au segment qui suit le segment (2) soit le segment (2').

Ce processus se poursuit jusqu'à la fin du fichier. (dans le cas schématisé ici, le fichier sera rétréci par rapport à l'original).

L'explication de cette technique a été trouvé à la section 2.1.5 WSOLA d'une thèse de doctorat nommée "Audio time-scale modification" écrite par David Dorran du Dublin Institute of Technology. Vous trouvez cette thèse sur le CD.

Voici un schéma simple illustrant le module:



3.7 Jouer son

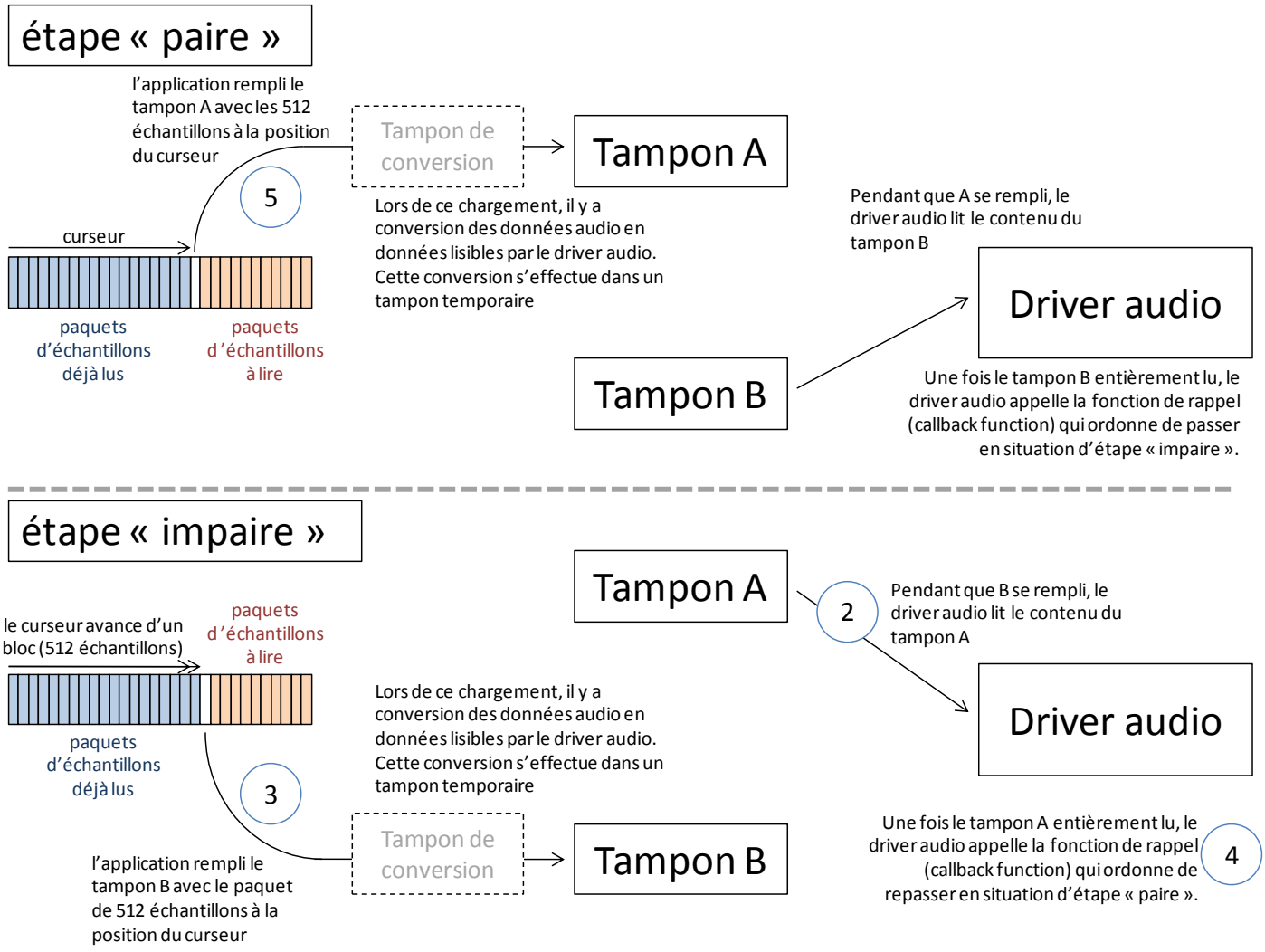
Le module jouer son consiste à ouvrir le fichier *.wav* et de jouer celui-ci, cela en utilisant les fonctions de la libraire *SDL_Audio*.

Une exécution typique de *SDL_Audio*: il demande deux tampons de la taille d'échantillons spécifié (512 pour nous) et les pré-remplis de silence. Ensuite SDL utilise la technique du "double buffering" (ce qui permet d'assurer un flux continu de données) en remplissant un tampon (grâce à la fonction de rappel) avec du son, pendant que le hardware audio lit l'autre tampon.

Le programme traite le son wave par bloc de 512 échantillons et créé deux tampons A et B de la même taille. Chaque tampon a une taille de 1ko (512 échantillons de 16 bits mono)

1. A et B sont pré-remplis par du silence
2. le hardware audio est déclenché pour jouer le contenu du tampon A
3. l'application est rappelée pour remplir B (avec les samples suivants)
4. le driver audio attend que A ait fini d'être lu et enclenche la lecture du tampon B
5. l'application est rappelée pour remplir A (avec les samples suivants)
6. en bouclant de cette manière jusqu'à ce que tous les échantillons soient lus.

Ci-dessous un schéma illustrant le "double buffering" de SDL_Audio :



4 GLOSSAIRE

-E-
échantillon : court segment de fichier audio numérique

-S-
SDL_Audio : librairie libre permettant de jouer des fichiers audio.

-T-
tampon : zone de mémoire vive pour stocker temporairement des données

-W-
WSOLA : algorithme *Wave-form Similarity-based Over-Lap Add* est une technique de modification de vitesse sans altérer la fréquence.

WAVE : format standard pour stocker de l'audio numérique

5 REFERENCES

Cahier des charges *S1*

Cahier de recette *S2*

lien explicant la structure d'un fichier son *.wav*
https://ccrma.stanford.edu/courses/422/projects/WaveFormat/

lien vers le site officiel d'SDL, expliquant le fonctionnement de SDL_Audio:
http://osdl.sourceforge.net/main/documentation/rendering/SDL-audio.html

lien sur lequel été trouvé le code source libre qui est utilisé dans le module *jouer son*
http://www.gnurou.org/writing/linuxmag/sdl/partie5

documentation officielle des fonctions de SDL_Audio
http://sdl.beuc.net/sdl.wiki/SDL_Audio

6 INDEX